# AI & ML

MODULE 2

SESSION 7

# Session Outline

- Data Preparation – in general
- Tasks in data preparation
- Basic data cleaning
- Handling outliers
- Handling Missing values
- Feature Scaling
- Categorical Variable encoding
- Train Test split

# Data Preparation

Transformation of raw data into a form more suitable for modelling.

all kinds of tasks and activities to detect and repair errors in the data.

**Data wrangling**

**Data cleaning**

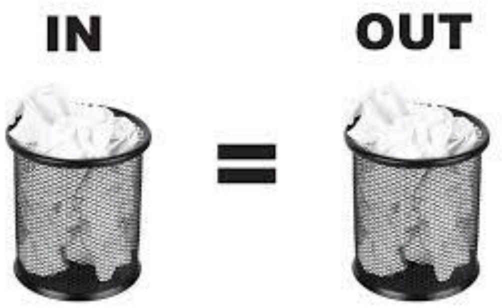
**Data pre-processing**

**Feature engineering**

**Data Preparation**

# Why data pre-processing?

***The performance of ML algorithm is only as good as the data used to train it***




- ML Algorithms expect numbers
- ML Algorithms have requirements
- Model performance depends on data

# How to choose data preparation techniques?

**BEFORE**

- Gather Data
- Discuss project with subject experts
- Select input & output variables.
- Review data
- Summarize and visualize data

→ Data Preparation →

**AFTER**

- Select performance metric for model.
- Select model evaluation procedure.
- Select algorithms to evaluate.
- Tune algorithm hyperparameters.

# General pre-processing tasks

**Data Cleaning**: Identifying and correcting mistakes or errors in the data.

**Data Transforms**: Changing the scale or distribution of variables.

**Feature Engineering**: Deriving new variables from available data.

**Feature Selection**: Identifying those input variables that are most relevant to the task.

**Dimensionality Reduction**: Creating compact projections of the data.

# Basic Data Cleaning

# Data Cleaning

**Basic Task**

**Identify and remove**

- Columns that contain single value
- Duplicating rows

# delete columns with a single unique value

```
from pandas import read_csv
# load the dataset
df = read_csv('oil-spill.csv', header=None)
print(df.shape)
# get number of unique values for each column
counts = df.nunique()
# record columns to delete
to_del = [i for i,v in enumerate(counts) if v == 1]
print(to_del)
# drop useless columns
df.drop(to_del, axis=1, inplace=True)
print(df.shape)
```

# delete rows of duplicate data from the dataset

```
from pandas import read_csv
# load the dataset
df = read_csv('iris.csv', header=None)
print(df.shape)
# delete duplicate rows
df.drop_duplicates(inplace=True)
print(df.shape)
```

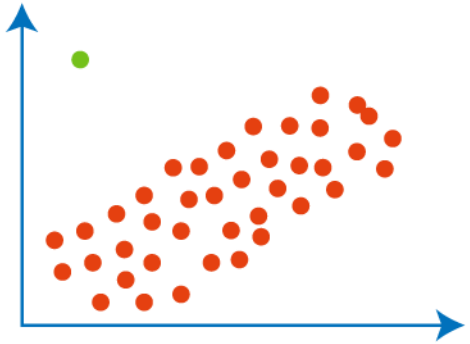
# Outlier Identification & Removal

# Outliers

**Observation unlike the other observations.**

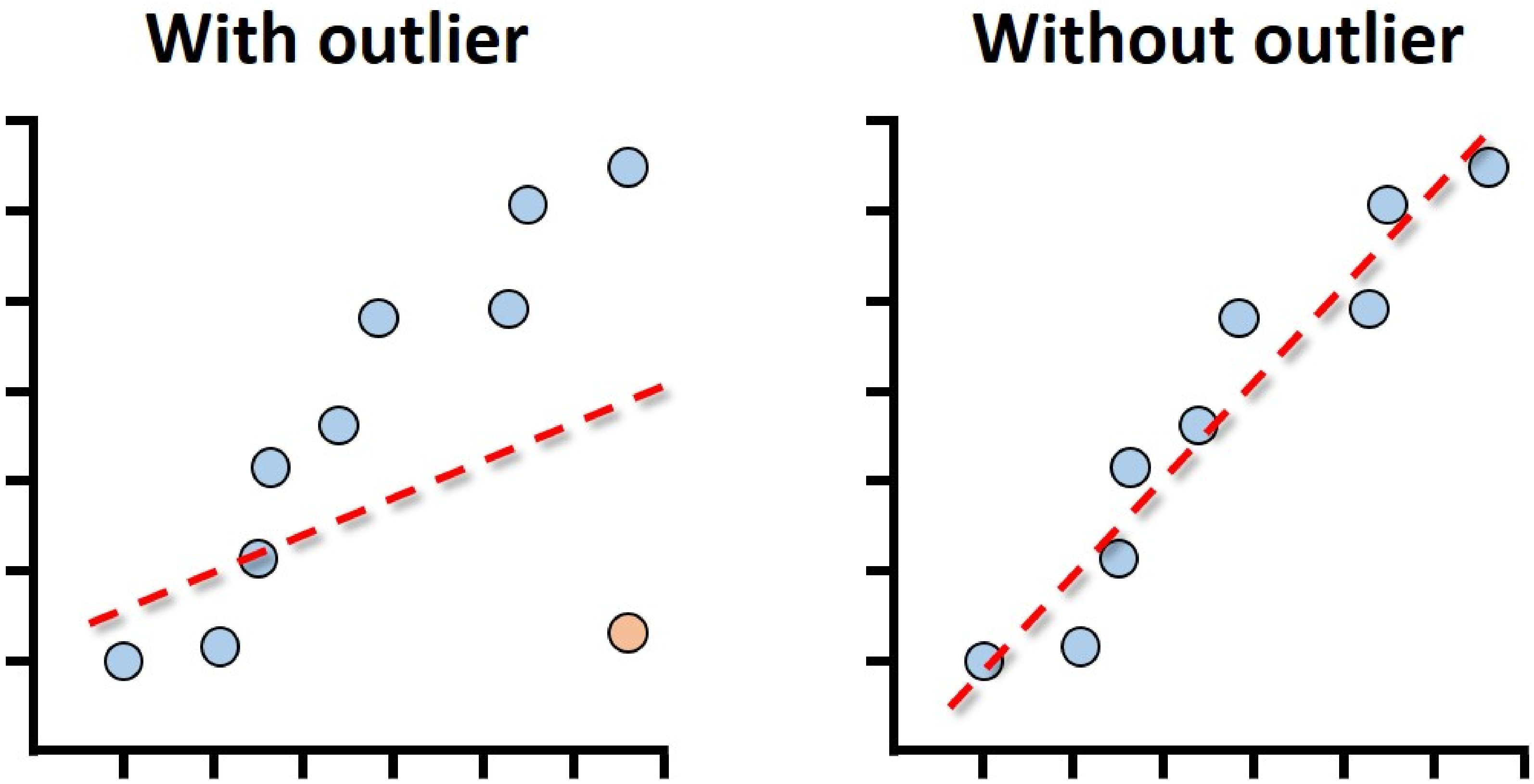
**Causes of outliers:**

1. Measurement or input error
2. Data corruption
3. True outlier observation



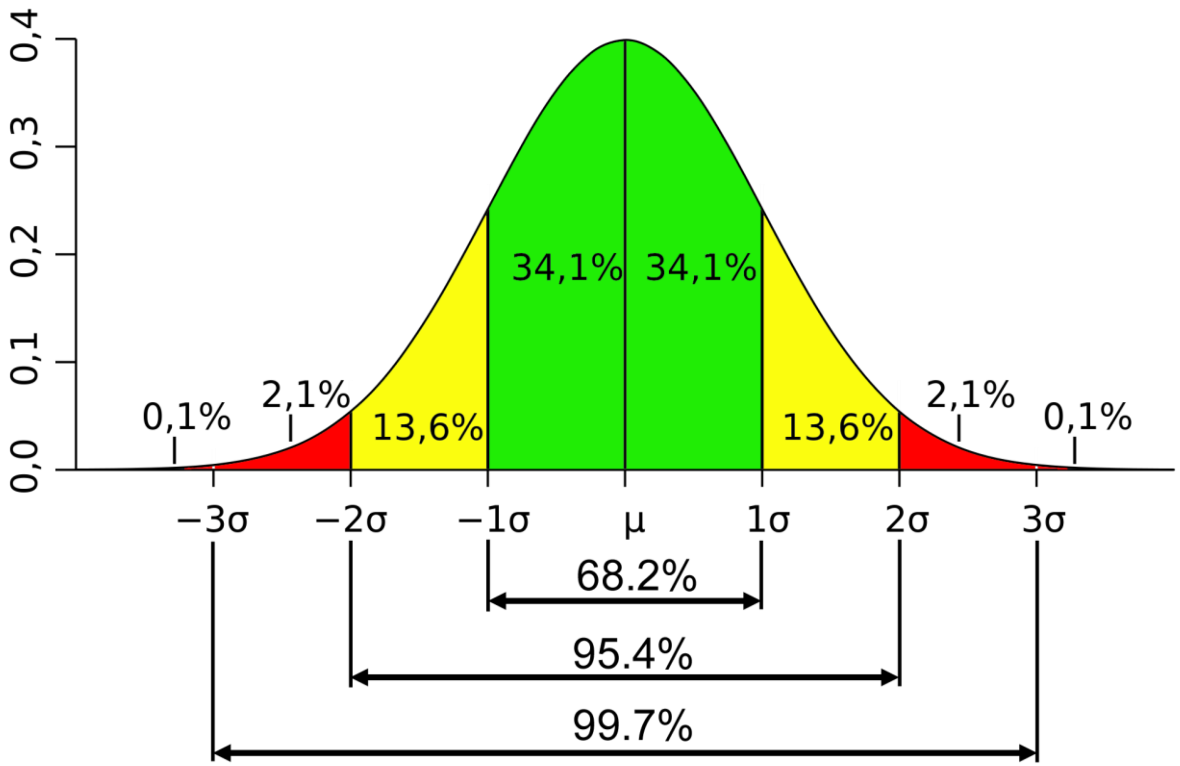
No precise way to define and identify an outlier in general.

# Prediction in presence of outliers



# Standard Deviation Method

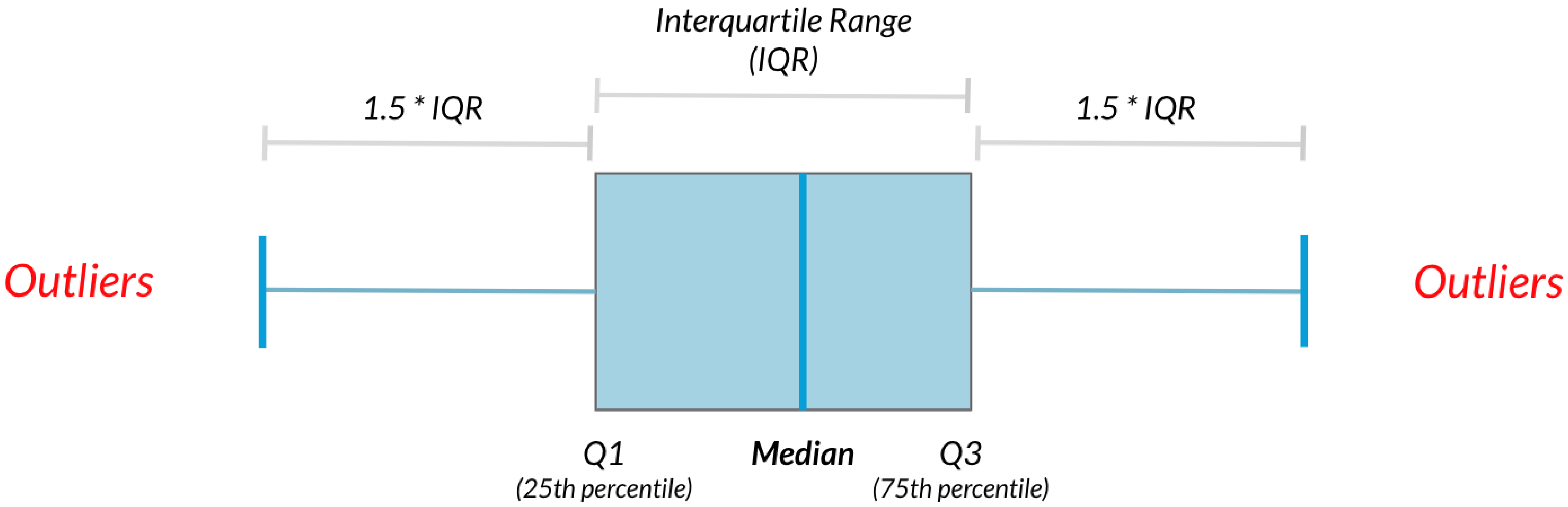
When distribution of values in sample is Gaussian or Gaussian-like



## *identify outliers with standard deviation*

```
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

# Interquartile Range Method



# IQR Calculation

123 156 180 190 267 290 300 322 342 365 395

Next identify the median.

123 156 180 190 267 290 300 322 342 365 395

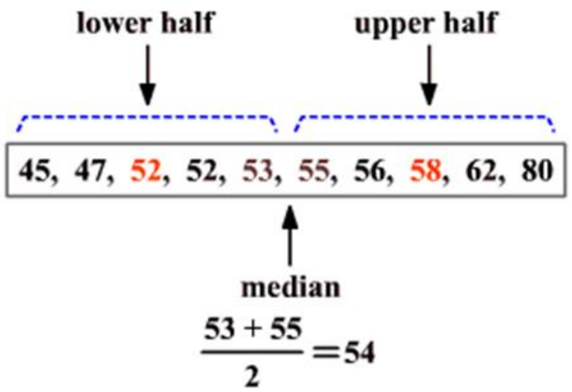
Identify the median for the lower and higher set of data

123 156 180 190 267 290 300 322 342 365 395

Lower quartile

Upper quartile

Interquartile range = 342-180 = 162mm



## *identify outliers with interquartile range*

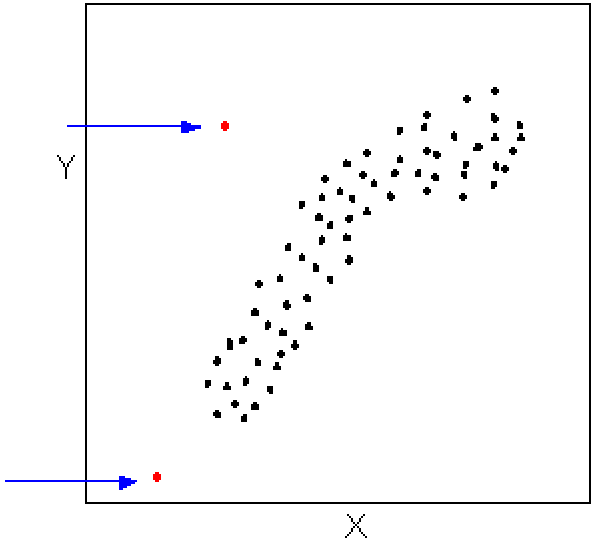
```
from numpy.random import seed
from numpy.random import randn
from numpy import percentile
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

# Automatic Outlier Detection

One class classification problem.

- to locate those examples that are far from the other examples in the multi-dimensional feature space.

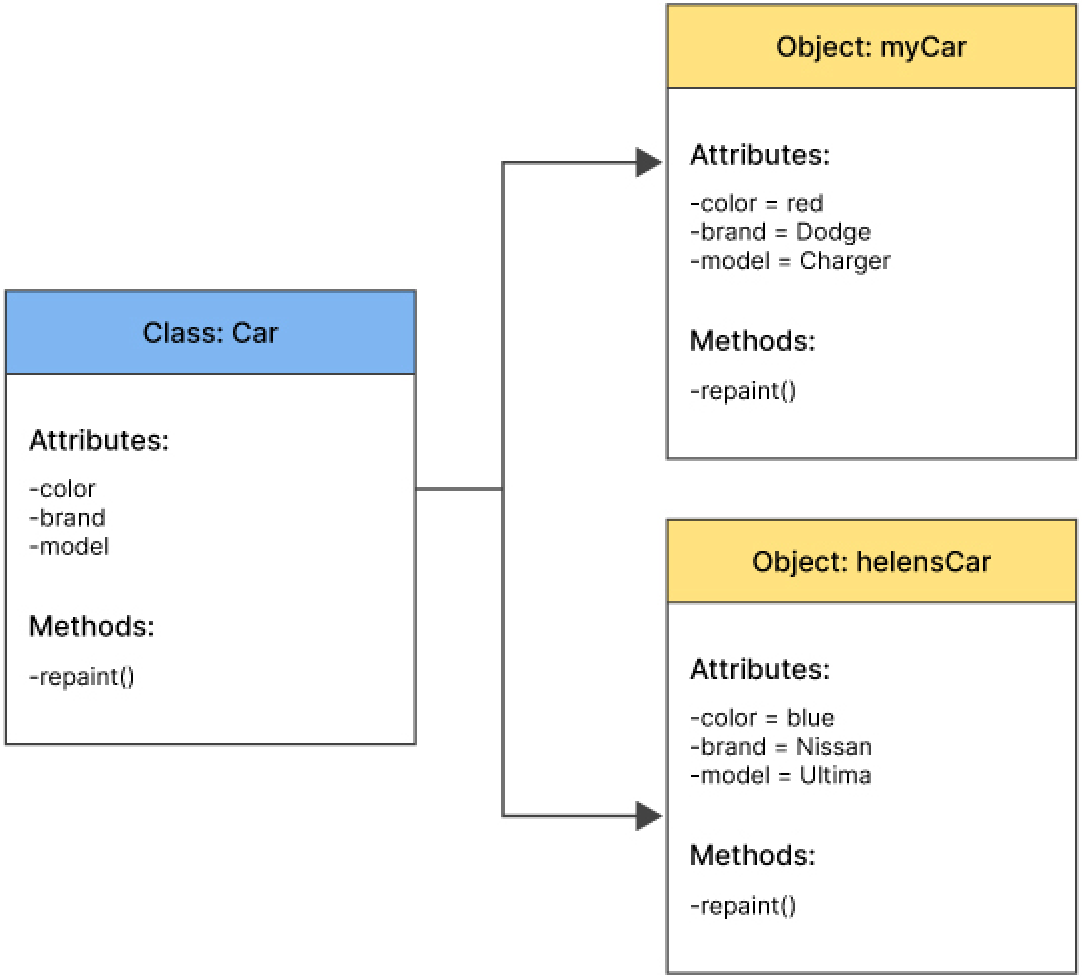
This can work well for feature spaces with low dimensionality (few features)



# Automatic outlier detection

```
from sklearn.neighbors import LocalOutlierFactor


lof = LocalOutlierFactor()

yhat = lof.fit_predict(X_train)

# select all rows that are not outliers

mask = yhat != -1

X_train, y_train = X_train[mask, :], y_train[mask]
```

# OOP


Class: Car
Attributes:
-color
-brand
-model
Methods:
-repaint()
Object: myCar
Attributes:
-color = red
-brand = Dodge
-model = Charger
Methods:
-repaint()
Object: helensCar
Attributes:
-color = blue
-brand = Nissan
-model = Ultima
Methods:
-repaint()

# Handling Missing Values

# Problem with missing values

- Having missing values in a dataset can cause errors with some machine learning algorithms.
- Many popular predictive models such as support vector machines, the glmnet, and neural networks, cannot tolerate any amount of missing values.
- The simplest strategy for handling missing data is to remove records that contain a missing value.

# Missing value imputation

1. Statistic imputation
   1. Mean
   2. Median
   3. Mode
   4. Constant
2. KNN Imputation
3. Iterative Imputation

# Statistical Imputation with SimpleImputer

from sklearn.impute import SimpleImputer

………

**# define imputer**

imputer = SimpleImputer(strategy='mean’) #other startegies 'median', 'most_frequent', 'constant'

...

**# fit on the dataset**

imputer.fit(X)

...

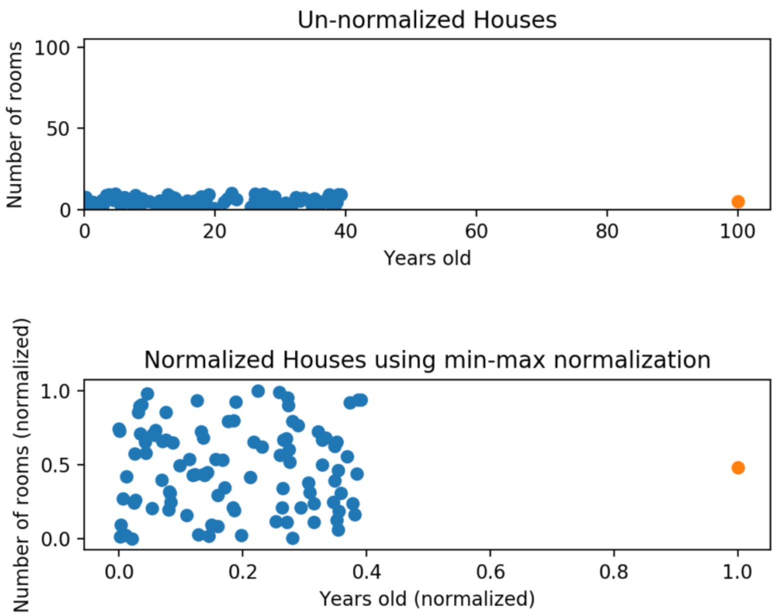
**# transform the dataset**

Xtrans = imputer.transform(X)

# Scaling Numerical Data

# Normalization

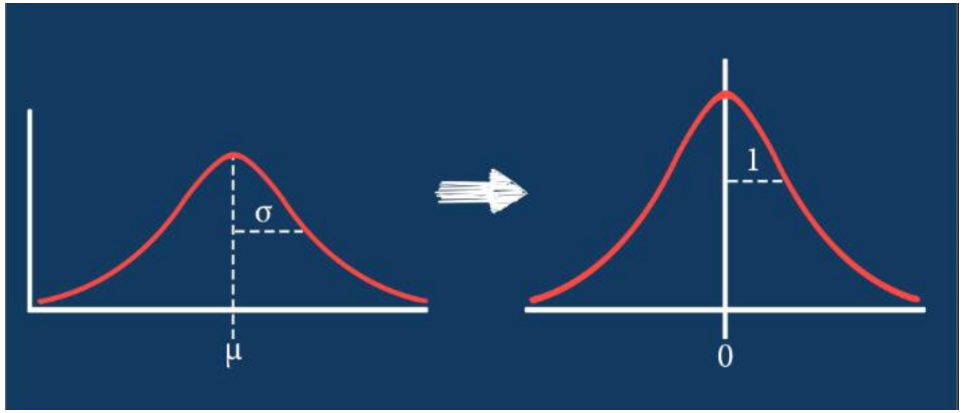
$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$



# example of a normalization

```
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler
# define data
data = asarray([[100, 0.001],
[8, 0.05],
[50, 0.005],
[88, 0.07],
[4, 0.1]])
print(data)
# define min max scaler
scaler = MinMaxScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```

# Standardization



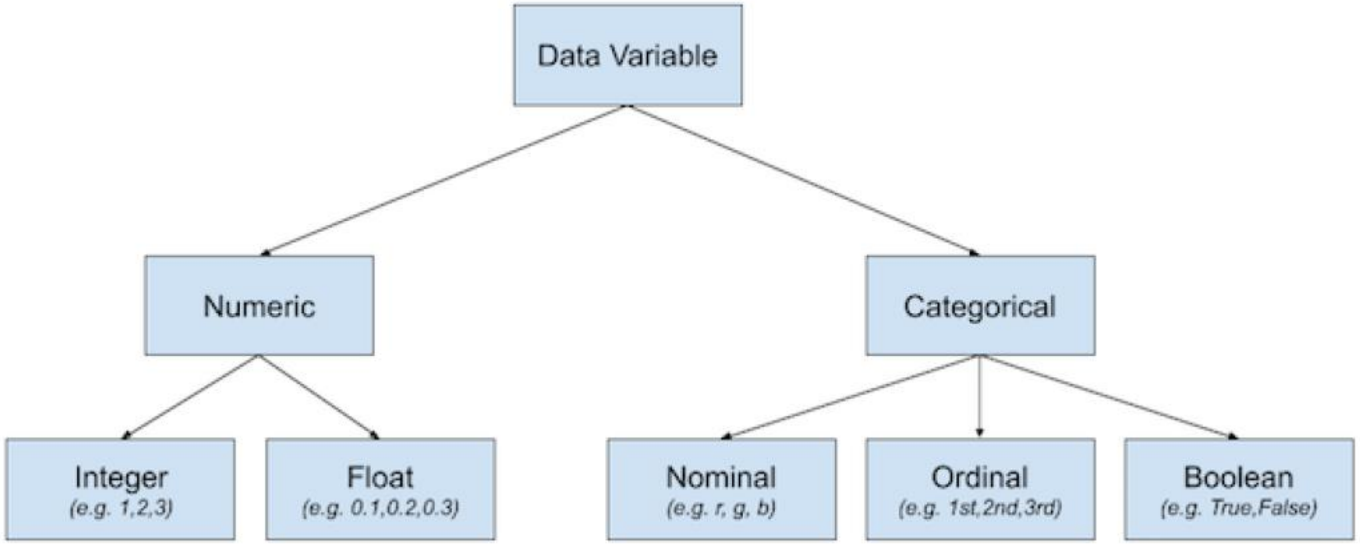

$$X' = \frac{X - \text{Mean}}{\text{Standard deviation}}$$

# example of a standardization

```
from numpy import asarray
from sklearn.preprocessing import StandardScaler
# define data
data = asarray([[100, 0.001],
[8, 0.05],
[50, 0.005],
[88, 0.07],
[4, 0.1]])
print(data)
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```

# Encode Categorical Data

# Overview of Data Variable Types



# Encoding Categorical Data

Ordinal Encoding

One Hot Encoding

Dummy Variable Encoding

# Ordinal Encoding vs One Hot Encoding

| color |
|---|
| red |
| green |
| blue |
| red |

| color |
|---|
| 0 |
| 1 |
| 2 |
| 0 |

| Color |
|---|
| Red |
| Red |
| Yellow |
| Green |
| Yellow |

| Red | Yellow | Green |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

# example of a ordinal encoding

```
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```

# example of a one hot encoding

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

# Dummy Variable Encoding

| id | X |
|---|---|
| 1 | a |
| 2 | c |
| 3 | a |
| 4 | b |
| 5 | a |
| 6 | c |
| 7 | c |
| 8 | b |

One-Hot Encoding

| id | X = a | X = b | X = c |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 |
| 5 | 1 | 0 | 0 |
| 6 | 0 | 0 | 1 |
| 7 | 0 | 0 | 1 |
| 8 | 0 | 1 | 0 |

Dummy Encoding

| id | X = a | X = b |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 0 | 0 |
| 3 | 1 | 0 |
| 4 | 0 | 1 |
| 5 | 1 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 0 | 1 |

# example of a dummy variable encoding

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(drop='first', sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

# Encoding Target Data

# Label Encoding

| SAFETY-LEVEL (TEXT) | SAFETY-LEVEL (NUMERICAL) |
|---|---|
| None | 0 |
| Low | 1 |
| Medium | 2 |
| High | 3 |
| Very-High | 4 |

# Label encode

```
from pandas import read_csv

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import OrdinalEncoder

# load the dataset

dataset = read_csv('breast-cancer.csv',
header=None)

# retrieve the array of data

data = dataset.values

# separate into input and output columns

X = data[:, :-1].astype(str)

y = data[:, -1].astype(str)

# ordinal encode input variables

ordinal_encoder = OrdinalEncoder()

X = ordinal_encoder.fit_transform(X)

# ordinal encode target variable

label_encoder = LabelEncoder()

y = label_encoder.fit_transform(y)

# summarize the transformed data

print('Input', X.shape)

print(X[:5, :])

print('Output', y.shape)

print(y[:5])
```

# Train Test Split

# Common Approach and its pitfall

1. ***Prepare Dataset***
2. ***Split Data***
3. ***Evaluate Models***

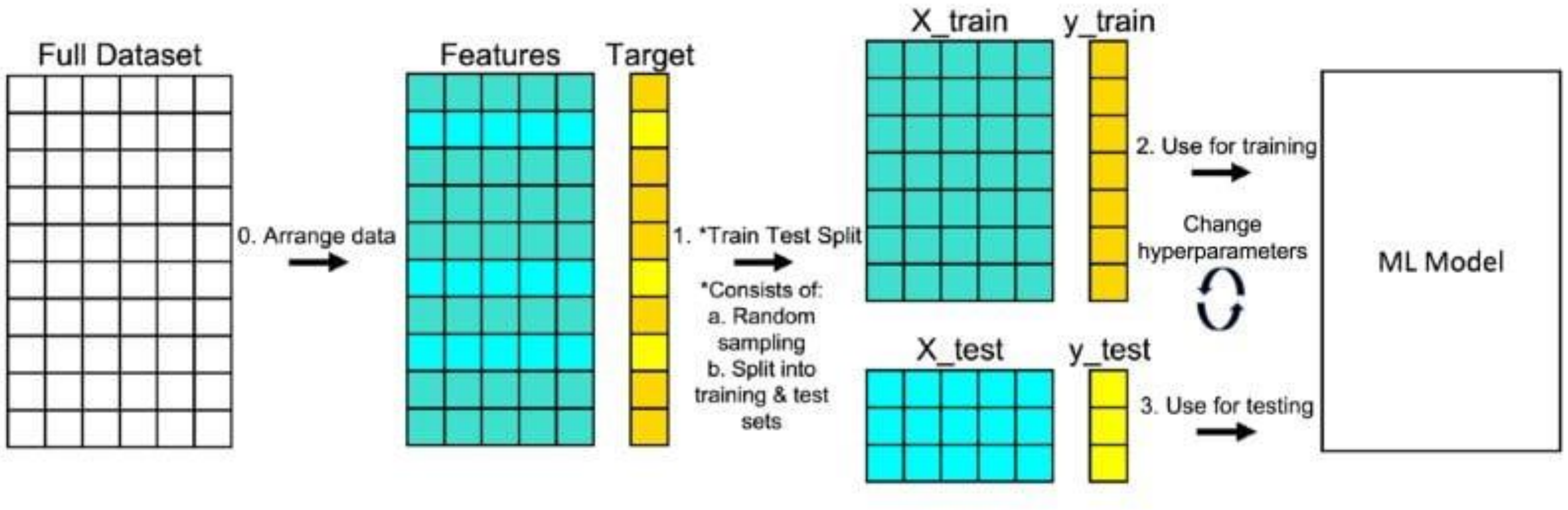
The manner in which data preparation techniques are applied matters.

We get data leakage by applying data preparation techniques to the entire dataset.

# To avoid data leakage

1. Split Data.
2. Fit Data Preparation on Training Dataset.
3. Apply Data Preparation to Train and Test Datasets.
4. Evaluate Models.

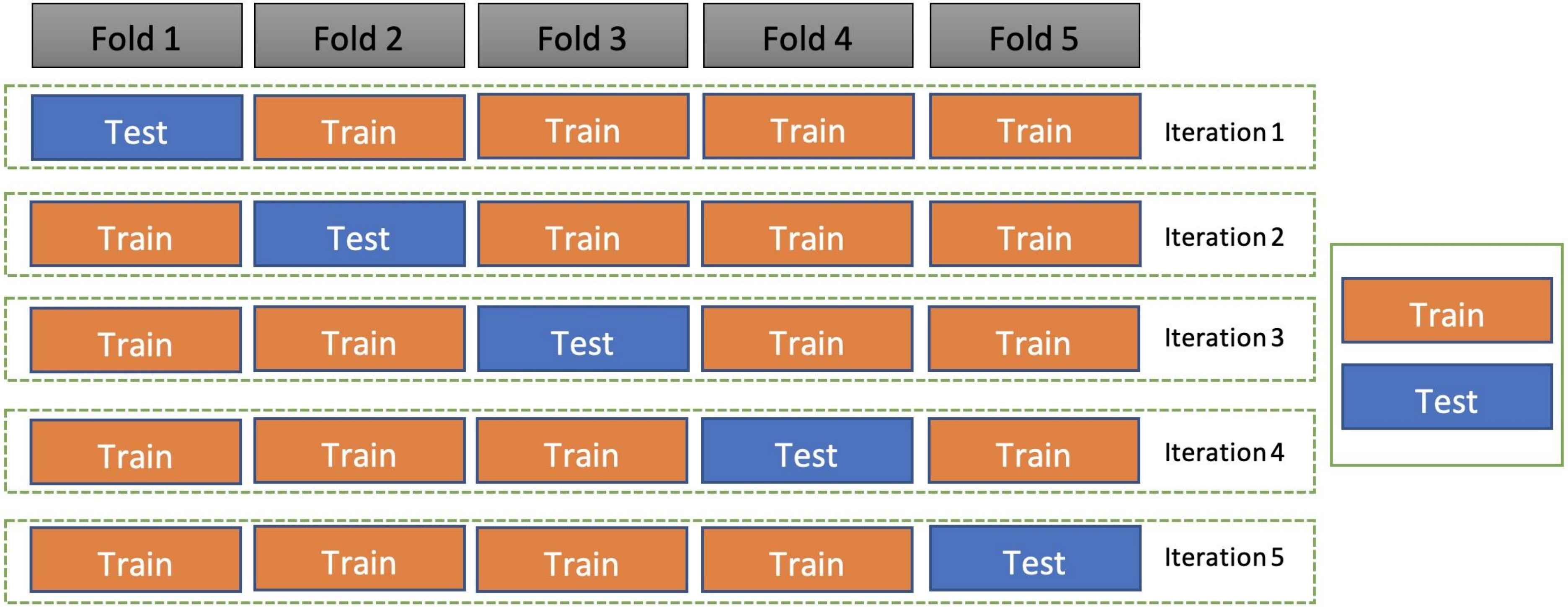
# Train-Test split



# Code

```
#Importing module for train test split

from sklearn.model_selection import train_test_split



#doing train test split

#test data = 33%; X =Features; y = target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
```

# K fold cross validation



# Reference

Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python

Available for download from

[Data Preparation for Machine Learning - Data Cleaning, Feature Selection, and Data - DOKUMEN.PUB]

# Dataset links

**Breast Cancer**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/breast-cancer.names

https://raw.githubusercontent.com/jbrownlee/Datasets/master/breast-cancer.csv

**Horse colic dataset**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/horse-colic.names

https://raw.githubusercontent.com/jbrownlee/Datasets/master/horse-colic.csv

**Diabetes dataset**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names

https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv

**Housing**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.names

https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv

**Iris**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv

**oil spill**

https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv

# Thank You!